NAG Library Function Document

nag_dormrq (f08ckc)

1 Purpose

nag_dormrq (f08ckc) multiplies a general real m by n matrix C by the real orthogonal matrix Q from an RQ factorization computed by nag_dgerqf (f08chc).

2 Specification

```
#include <nag.h>
#include <nagf08.h>
```

```
void nag_dormrq (Nag_OrderType order, Nag_SideType side,
     Nag_TransType trans, Integer m, Integer n, Integer k, double a[],
     Integer pda, const double tau[], double c[], Integer pdc,
     NagError *fail)
```

3 Description

nag_dormrq (f08ckc) is intended to be used following a call to nag_dgerqf (f08chc), which performs an RQ factorization of a real matrix A and represents the orthogonal matrix Q as a product of elementary reflectors.

This function may be used to form one of the matrix products

$$QC, \quad Q^{\mathrm{T}}C, \quad CQ, \quad CQ^{\mathrm{T}},$$

overwriting the result on C, which may be any real rectangular m by n matrix.

A common application of this function is in solving underdetermined linear least squares problems, as described in the f08 Chapter Introduction, and illustrated in Section 10 in nag_dgerqf (f08chc).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **side** – Nag_SideType *Input*

On entry: indicates how Q or Q^{T} is to be applied to C.

side = Nag_LeftSide
Q or Q^{T} is applied to C from the left.

side = Nag_RightSide
Q or Q^{T} is applied to C from the right.

Constraint: **side** = Nag_LeftSide or Nag_RightSide.

3: **trans** – Nag_TransType *Input*

On entry: indicates whether Q or Q^{T} is to be applied to C.

trans = Nag_NoTrans
Q is applied to C.

trans = Nag_Trans
Q^{T} is applied to C.

Constraint: **trans** = Nag_NoTrans or Nag_Trans.

4: **m** – Integer *Input*

On entry: m, the number of rows of the matrix C.

Constraint: $\mathbf{m} \geq 0$.

5: **n** – Integer *Input*

On entry: n, the number of columns of the matrix C.

Constraint: $\mathbf{n} \geq 0$.

6: **k** – Integer *Input*

On entry: k, the number of elementary reflectors whose product defines the matrix Q.

Constraints:

if **side** = Nag_LeftSide, $\mathbf{m} \geq \mathbf{k} \geq 0$;
if **side** = Nag_RightSide, $\mathbf{n} \geq \mathbf{k} \geq 0$.

7: **a**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least

$\max(1, \mathbf{pda} \times \mathbf{m})$ when **side** = Nag_LeftSide and **order** = Nag_ColMajor;
$\max(1, \mathbf{k} \times \mathbf{pda})$ when **side** = Nag_LeftSide and **order** = Nag_RowMajor;
$\max(1, \mathbf{pda} \times \mathbf{n})$ when **side** = Nag_RightSide and **order** = Nag_ColMajor;
$\max(1, \mathbf{k} \times \mathbf{pda})$ when **side** = Nag_RightSide and **order** = Nag_RowMajor.

The (i, j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the ith row of **a** must contain the vector which defines the elementary reflector H_i, for $i = 1, 2, \ldots, k$, as returned by nag_dgerqf (f08chc).

On exit: is modified by nag_dormrq (f08ckc) but restored on exit.

8: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pda} \geq \max(1, \mathbf{k})$;
if **order** = Nag_RowMajor,

if **side** = Nag_LeftSide, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, $\mathbf{pda} \geq \max(1, \mathbf{n})$..

9: **tau**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **tau** must be at least $\max(1, \mathbf{k})$.

On entry: **tau**$[i-1]$ must contain the scalar factor of the elementary reflector H_i, as returned by nag_dgerqf (f08chc).

10: **c**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **c** must be at least

$\max(1, \mathbf{pdc} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdc})$ when **order** = Nag_RowMajor.

The (i,j)th element of the matrix C is stored in

c$[(j-1) \times \mathbf{pdc} + i - 1]$ when **order** = Nag_ColMajor;
c$[(i-1) \times \mathbf{pdc} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the m by n matrix C.

On exit: **c** is overwritten by QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T} as specified by **side** and **trans**.

11: **pdc** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdc} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pdc} \geq \max(1, \mathbf{n})$.

12: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_3

On entry, **side** = ⟨*value*⟩, **m** = ⟨*value*⟩, **n** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, $\mathbf{m} \geq \mathbf{k} \geq 0$;
if **side** = Nag_RightSide, $\mathbf{n} \geq \mathbf{k} \geq 0$.

On entry, **side** = ⟨*value*⟩, **pda** = ⟨*value*⟩, **m** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, $\mathbf{pda} \geq \max(1, \mathbf{n})$.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 0$.

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **pda** $= \langle value \rangle$.
Constraint: **pda** > 0.

On entry, **pdc** $= \langle value \rangle$.
Constraint: **pdc** > 0.

NE_INT_2

On entry, **pda** $= \langle value \rangle$ and **k** $= \langle value \rangle$.
Constraint: **pda** $\geq \max(1, \mathbf{k})$.

On entry, **pdc** $= \langle value \rangle$ and **m** $= \langle value \rangle$.
Constraint: **pdc** $\geq \max(1, \mathbf{m})$.

On entry, **pdc** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdc** $\geq \max(1, \mathbf{n})$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

The computed result differs from the exact result by a matrix E such that

$$\|E\|_2 = O\epsilon\|C\|_2$$

where ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_dormrq (f08ckc) is not threaded by NAG in any implementation.

nag_dormrq (f08ckc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations is approximately $2nk(2m - k)$ if **side** = Nag_LeftSide and $2mk(2n - k)$ if **side** = Nag_RightSide.

The complex analogue of this function is nag_zunmrq (f08cxc).

10 Example

See Section 10 in nag_dgerqf (f08chc).